MS SQL DML Triggers

DML тригерите са обекти, които са обвързани с конкретна таблица. Те представляват съвкупност от операции, които се изпълняват автоматично при настъпване на определено събитие (INSERT, UPDATE, DELETE). Има два вида тригери – AFTER (изпълняват се след настъпване на събитието) и INSTEAD OF (изпълняват се вместо командата при настъпване на събитието).

За тригерите са дефинирани две временни таблици, които съдържат информация за промените.

DML команда	Временна таблица	Описание
INSERT	***Inserted***	Съдържа данните, които току що са били вмъкнати
DELETE	***Deleted***	Съдържа данните, които току що са били изтрити
UPDATE	***Inserted***	Съдържа копие на данните след обновяването
	Deleted	Съдържа копие на данните преди обновяването

Тригерите са подходящи за:

- запазване информация за това кой потребител и кога е направил промени по дадени данни (потребител, дата на промяна)
- архивиране на данни (напр. преместване на неактивни поръчки)
- отменяне на операции
- проверка на състоянието на данните преди и след модификация
- показване на потребителски съобщения при изпълнение на команда

AFTER тригери

Изпълняват се след като дадена модификация (INSERT, UPDATE или DELETE) завърши успешно. Ограниченията, определени върху таблицата, се проверяват преди да се изпълни командата INSERT, UPDATE или DELETE и ако тези ограничения не се удовлетворяват, то тригерът не се изпълнява. AFTER тригерите могат да се създават само върху таблици. За дадена команда се позволява множество от AFTER тригери.

INSTEAD OF тригери

Изпълняват се вместо модификацията с INSERT, UPDATE или DELETE преди да бъдат проверени ограниченията върху таблицата. INSTEAD OF тригери могат да се създават както за таблици, така и за изгледи (***views***). За дадена таблица и дадено действие (INSERT, UPDATE или DELETE) се позволява само един INSTEAD OF тригер.

Ред на изпълнение на операциите при наличие на тригери

1. Извиква се командата INSERT, UPDATE или DELETE.
2. Ако е дефиниран INSTEAD OF тригер, то той се изпълнява вместо операцията.
3. Проверява се дали модификациите са съобразени с дефинираните ограничения. Ако ограниченията не са спазени, възниква грешка и операцията се прекратява.
4. Изпълняват се AFTER тригерите. Първият и последният тригер за изпълнение могат да се специфицират изрично. Всички останали се изпълняват в неопределен ред.
5. Транзакцията се затвърждава (COMMIT).

В MS SQL Server тригерът се изпълнява веднъж за извикване на INSERT, UPDATE или DELETE. Възможно е една такава команда да добавя/променя/изтрива множество кортежи (например INSERT SELECT). Всички променени кортежи имат копия в таблиците ***Inserted*** и/или ***Deleted*** в зависимост от вида на операцията.

Забележка: *В MS SQL Server е важно тригерите да се дефинират така, че да работят коректно както при промяна на един, така и на много кортежи. В Oracle и DB2 тригерът може да се изпълнява за всеки кортеж по отделно или наведнъж за цялата операция.*

За проверка на броя на засегнатите от операцията редове може да се използва вградената функция @@ROWCOUNT, която връща броя на вмъкнатите/променените/изтритите редове.

Синтаксис

```
CREATE TRIGGER trigger_name ON table_name

AFTER | INSTEAD OF action

AS

…
```

Процедурни конструкции в MS SQL Server (Transact-SQL)

- **Деклариране на променливи**

Променливи в Transact-SQL се декларират чрез запазената дума ***DECLARE.***Идентификаторите на променливи започват с ***@.*** След идентификатора се задава типът на променливата, който може да бъде и релация.

```
DECLARE @intValue INT;

DECLARE @decimalValue DECIMAL(6,2);
```

```
DECLARE @firstName VARCHAR(32), @lastName VARCHAR(32);

DECLARE @digits TABLE ([Value] INT);
```

Присвояване на стойност се извършва с оператора ***SET.***

```
SET @intValue = 7;
```

За вмъкване на резултат в променлива от тип релация може да се използва ***INSERT.***

```
INSERT INTO @digits([Value])
SELECT 0 UNION
SELECT 1 UNION
SELECT 2 UNION
SELECT 3 UNION
SELECT 4 UNION
SELECT 5 UNION
SELECT 6 UNION
SELECT 7 UNION
SELECT 8 UNION
SELECT 9;
```

- ***BEGIN… END***

Използва се за групиране на оператори.

- ***IF… ELSE***

Условен оператор, за който начинът на изпълнение е аналогичен на този в езиците за програмиране. Оценява се булевото условие и ходът на изпълнение се разклонява в зависимост от истинността на условието.

```
IF (@@ERROR <> 0)
   BEGIN
         PRINT ERROR_MESSAGE();
   END
ELSE
   BEGIN
         PRINT 'No error occurred.';
   END
```

- ***WHILE***

Конструкция за цикъл – тялото се изпълнява докато условието е вярно – аналогично на циклите в езиците за програмиране.

```
DECLARE @Counter int;
SET @Counter = 1;
WHILE (@Counter <= 10)
BEGIN
   PRINT @Counter;
   SET @Counter = @Counter + 1;
END
```

- **Обработка на грешки**

Информация за различни характеристики на възникналите грешки може да се получи със следните функции:

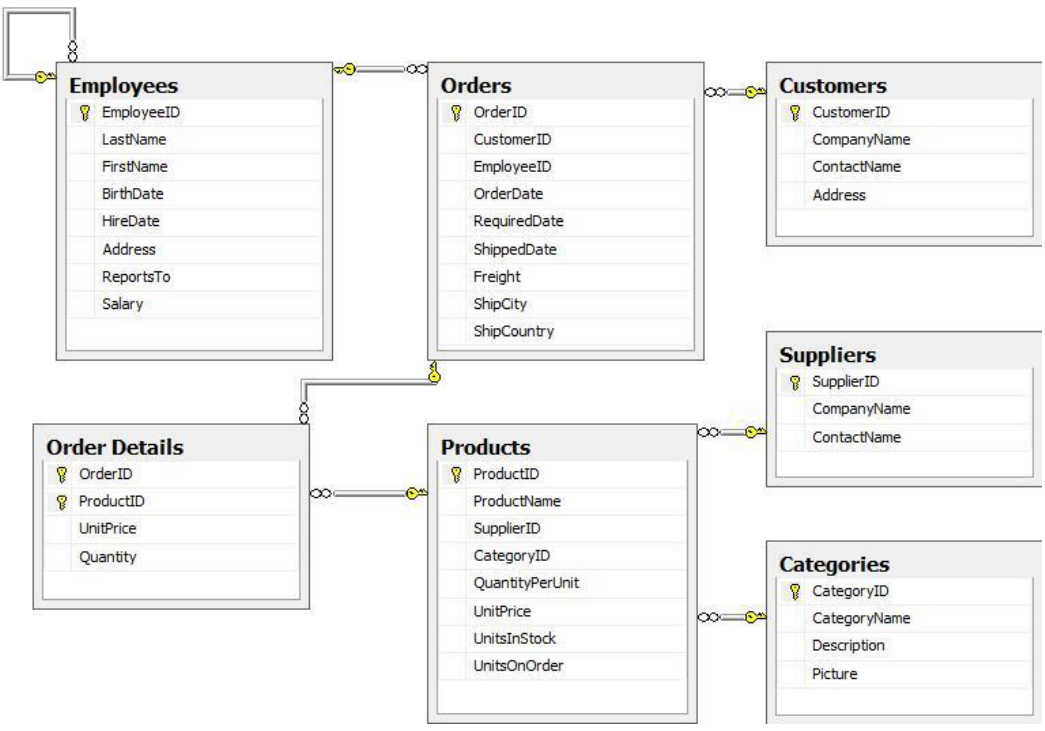
ERROR_LINE() – ред, на който е възникнала грешката
ERROR_NUMBER() – номер на грешката (за системни грешки е число < 50000; на потребителски дефинираните грешки трябва да се задава код на грешката >= 50000)
ERROR_MESSAGE() – съобщение на грешката

Потребителски дефинирана грешка може да се зададе с функцията ***RAISERROR***.

@@ERROR съдържа код на възникналата грешка. Логиката при обработване на грешки с конструкция ***IF @@ERROR <> 0*** се препоръчва да се избягва. За предпочитане е да се използва

```
BEGIN TRY

--код, който може да причини грешка

END TRY

BEGIN CATCH

--обработка на грешката при нейното възникване

END CATCH
```

Задачи



Използвайте базата данни Northwind.

1. Да се създаде таблица, която за всеки служител пази следната информация:
   - ✓ идентификатор на служителя (не трябва да съществува идентификатор, който не присъства в таблицата Employees)
   - ✓ брой обслужени от него поръчки (0 по подразбиране)
   - ✓ дата на последната обслужена от служителя поръчка
   - ✓ последната дата на обновяване на информацията за служителя в таблицата

   EmployeeInfo(EmployeeID, NumOrdersServed, LatestOrderDate, LatestUpdateDate)

2. Да се актуализира информацията за служителите в тази таблица спрямо текущото съдържание на Orders.
3. Да се създаде AFTER тригер, който след всяко изпълнение на INSERT в таблицата Orders подновява информацията за служителите в новата таблица.
4. Добавете атрибут статус на служителя към релацията EmployeeInfo. По подразбиране нека статусът е 'Active'.
5. Създайте AFTER тригер, който след вмъкване на служител в таблицата Employees създава запис за служителя в EmployeeInfo.
6. Променете AFTER тригера от задача 3 така, че след добавяне на поръчки чрез INSERT в Orders да сменя статуса на служителя, направил най-много поръчки, на 'Superactive'.
7. Създайте INSTEAD OF тригер, който при опит за смяна на датата на съществуваща поръчка връща грешка, а в противен случай извършва успешно операцията.

Примерно решение на задача 3.

```
CREATE TRIGGER TrgAfterOrderInsert ON Orders
AFTER INSERT
AS
BEGIN
	--update existing employees
	UPDATE e
	SET NumOrdersServed = NumOrdersServed +
				(SELECT COUNT(OrderID)
				 FROM Inserted
				 WHERE e.EmployeeID = EmployeeID),
		LatestOrderDate =
			(SELECT CASE
					WHEN MAX(OrderDate) >
						LatestOrderDate THEN MAX(OrderDate)
					ELSE LatestOrderDate
				END
			 FROM Inserted
			 WHERE e.EmployeeID = EmployeeID),
		LatestUpdateDate = GETDATE()
	FROM EmployeeInfo e
	WHERE e.EmployeeID IN (SELECT EmployeeID FROM Inserted);
	--update new employees
	IF EXISTS (SELECT *
				FROM EmployeeInfo e
				RIGHT OUTER JOIN Inserted i
				ON e.EmployeeID = i.EmployeeID
				WHERE e.EmployeeID IS NULL)
	BEGIN
		INSERT INTO EmployeeInfo
		SELECT i.EmployeeID, COUNT(i.OrderID),
			MAX(i.OrderDate), GETDATE()
		FROM Inserted i
		LEFT OUTER JOIN EmployeeInfo e
		ON i.EmployeeID = e.EmployeeID
		WHERE e.EmployeeID IS NULL
		GROUP BY i.EmployeeID;
	END
END
```